



Requirement Documentation

Version 1.0.0

Vocab system

Parisa Ebrahemian

Shayan Ahmadi

Definition

This project follows the MVC architecture to separate business logic, user interface, and data handling. This structure improves scalability, maintainability, and testability.

This application is an **SSR (Server Side Rendering)** program that renders the entire view on the backend.

MVC Components

Model:

This section is responsible for defining the database models and logic. For this project, three databases have been considered:

- **User:** Contains information such as email, hashed password, last login, user role (which can be either a regular user or an admin), the number of words the user has created and the profile Images that the user uploads.
- **Word:** Contains details such as the English equivalent of the word, the Persian equivalent, the pronunciation, an example sentence using the word, and the user who created it.
- **Activity:** Includes the user who has logged in along with the exact date and time of the login.

All models include the fields **id**, **creation date**, and **update date**.

View:

This section handles the display of data in the correct locations and manages the user interface. It includes the following pages:

- addWord.ejs
- crm.ejs
- edit-pass.ejs
- home.ejs
- login.ejs
- new-pass.ejs
- profile.ejs
- register.ejs
- send-code.ejs
- submit-code.ejs

controller

This section is responsible for processing requests, managing the interaction between the **Model** and **View**, and handling server routing.

Additionally, it manages user authentication, CAPTCHA verification, and sending verification codes to users' emails for password recovery.

A separate controller has been defined for each entity in the **Model**, which includes the following files:

- page.controller.js
- user.controller.js
- word.controller.js

Other Folders and Files

Utils

This folder contains error handlers, and configuration scripts. It includes the following files:

- errorHandler.js
- mongo.config.js
- multer.config.js
- passport.config.js

Router

All available server routes are defined in this folder. A separate router is created for each model, where the routes related to that model are defined and managed.

Finally, there is a central router that integrates all individual routers, configuring their routes and ultimately introducing all available routes to the server.

- all.router.js
- page.router.js
- user.router.js
- word.router.js

view/components

This section contains all reusable UI components that may be used multiple times throughout the application.

view/public

his folder contains all static files, including images, stylesheets, and user-uploaded profile pictures. These files are organized into the following subfolders:

- img/ – Stores general images.
- styles/ – Contains CSS stylesheets.
- uploads/ – Holds profile pictures uploaded by users.

Folder & File Structure

source/

```
|—— controller/
|   |—— page.controller.js
|   |—— user.controller.js
|   |—— word.controller.js
|—— model/
|   |—— activity.model.js
|   |—— user.model.js
|   |—— word.model.js
|—— node_modules/
|   |—— <dependencies_instalation>
|—— router/
|   |—— all.router.js
```

```
|   |—— page.router.js
|   |—— user.router.js
|   |—— word.router.js
|—— utils/
|   |—— errorHandler.js
|   |—— mongo.config.js
|   |—— multer.config.js
|   |—— passport.config.js
|—— view/
|   |—— components/
|   |   |—— member.ejs
|   |   |—— nav.ejs
|   |   |—— word.ejs
|   |   |—— wordCard.ejs
|   |   |—— wordList.ejs
|   |—— public/
|   |   |—— img/
|   |   |   |—— <images_icons>
|   |   |—— styles/
|   |   |—— crm.css
```

- | | | | — home.css
- | | | | — list.css
- | | | | — login.css
- | | | | — new-pass.css
- | | | | — profile.css
- | | | | — register.css
- | | | | — send-code.css
- | | | | — submit-code.css
- | | | — uploads/
- | | | — <user_profile_storage>
- | | — addWord.ejs
- | | — crm.ejs
- | | — edit-pass.ejs
- | | — home.ejs
- | | — login.ejs
- | | — new-pass.ejs
- | | — profile.ejs
- | | — register.ejs
- | | — send-code.ejs
- | | — submit-code.ejs

|—— .env

|—— .gitignore

|—— package-lock.json

|—— package.json

|—— server.js

Dependencies and Tools

- Express.js: Main framework for server-side development
- Mongoose: ORM for managing MongoDB database
- EJS: Template engine for rendering dynamic views
- Bcrypt: Hashes passwords for secure storage.
- cookie-parser: Parses cookies in HTTP requests.
- Dotenv: Loads environment variables from a .env file.
- express-flash: Displays flash messages in the application.
- express-recaptcha: Integrates Google reCAPTCHA for bot protection.
- express-session: Manages user sessions.
- Multer: Handles file uploads.
- Nodemailer: Sends emails, used for verification and password recovery.
- Nodemon: Automatically restarts the server during development.
- Passport: Authentication middleware for handling user logins.
- passport-local: Implements local authentication using username and password.

Data Flow Documentation

User Requests and Page Routing

- Users navigate through the web application via various endpoints handled by Express.js.
- The routing is managed in `page.controller.js`, `user.controller.js`, and `word.controller.js`.

Page Flow

1. **Main Redirect (/):** If the user is authenticated, they are redirected to the home page; otherwise, they are taken to the login page.
2. **Authentication Pages:**
 - `/login-page`: Displays the login form with reCAPTCHA verification.
 - `/signup-page`: Displays the registration form with reCAPTCHA verification.
 - `/logout`: Ends the user session and redirects to the login page.
3. **User Dashboard and Profile:**
 - `/home-page`: Displays user-related words and activities.
 - `/profile-page`: Shows user details and their added words.
 - `/list-page`: Displays a list of words added by the user.
 - `/addword-page`: Renders the form for adding new words.
 - `/updateword-page`: Updates an existing word's details.
4. **Password Recovery:**

- /recoverpass-page: Requests email for password recovery.
- /submitcode-page: Verifies OTP sent via email.
- /newpass-page: Allows resetting the password.
- /editpass-page: Enables logged-in users to change their password.

5. Admin Dashboard:

- /admin-page: Available only for admin users. Displays user activity and statistics.

User Authentication Flow

- **Signup:** User provides email and password → Validations are performed → Data is stored in userModel.
- **Login:** Credentials are verified against userModel → Successful login stores session data → Redirects to home-page.
- **Logout:** User session is destroyed, redirecting them to login-page.

Word Management Flow

- **Add Word (/word/add):** User submits a new word → Data is stored in wordModel → User's word count is updated.
- **Update Word (/word/update):** User modifies an existing word → Updates wordModel.
- **Delete Word (/word/delete):** Removes a word from wordModel.
- **Search Word (/word/search):** Queries the database for matching words.

Password Recovery Flow

- User requests password reset → OTP is sent via email using Nodemailer.
- User submits OTP → Verified against stored values.
- If valid, user is redirected to newpass-page to reset the password.

Admin Flow

- Admin accesses /admin-page → System retrieves user activity and statistics from activityModel.
- Data is processed and visualized on the dashboard.
- Admin can add new users with admin privileges.

Middleware and Security

- **reCAPTCHA Verification:** Prevents bots from accessing login and signup.
- **Passport Authentication:** Secures user login with session management.
- **Session Management:** Ensures user state is maintained across requests.
- **Password Hashing:** Uses bcrypt for securely storing passwords.
- **Cookie Handling:** Stores session and verification tokens for password recovery.